# In the United States Patent and Trademark Office

In re the Application of:

| | | |
|---|---|---|
| Steven Edward Atkin | ) | |
| Serial Number: 09/838,377 | ) | Group: 2122 |
| Docket Number: AUS920010277US1 | ) | Examiner: Andre R. Fowlkes |
| Filed on: 04/19/2001 | ) | |
| For: "Bi-Directional Display" | ) | |

**AFFIDAVIT UNDER 37 CFR §1.132**

I, STEVEN EDWARD ATKIN, Ph.D., declare as follows:

1. I am the inventor and applicant of the invention entitled "Bi-Directional Display", disclosed and claimed in U.S. Patent Application Serial No. 09/838,377 filed on April 19, 2001.

2. The invention described and claimed in said application was conceived by me prior to October 4, 2000, as evidenced by the following facts which are of my own knowledge.

(a) The paper cited by the Examiner of said patent application, entitled "Implementations of Bidirectional Reordering Algorithms", Florida Tech Technical Report CS-2000-1, by Steven Atkin and Ryan Stansifer was authored by me. During my attendance at Florida Institute of Technology, Dr. Stansifer was my faculty advisor, and as such, acted in an advisory, reviewer and editor role only. This Technical Report was written during the development of my invention as claimed in said Patent Application. The Technical Report number was assigned upon my request on October 4, 2000, in advance of completion of the actual report, as the policy of the Technical Report Library was to assign report numbers upon submission of a title, author name, and abstract.

(b) I conceived of my invention prior to October 4, 2000, as evidenced by Appendix A to this affidavit, which includes several code files dated in June of 2000. These early prototypes of my invention were developed in order to test and prove the validity of the concept, some of the results of which were provided in the disclosure of the Patent Application.

(c) Following completion of verification testing and prototype code development, a disclosure was submitted to inside patent counsel for my employer, International Business Machines, on January 26, 2001. Shortly thereafter, I was notified that the disclosure had been selected for patenting, and I was contacted by outside patent counsel Robert Frantz. This effort resulted in the filing of the Patent Application on April 19, 2001.

3. As such, I am the sole inventor of the invention described and claimed in the Patent Application. Dr. Stansifer provided advisory input only to the cited Technical Report, and thus is only a co-author, but not a co-inventor. Further, I conceived of and prototyped the invention disclosed and claimed in the Patent Application prior to October 4, 2000.

[illegible]

Steven Edward Atkin

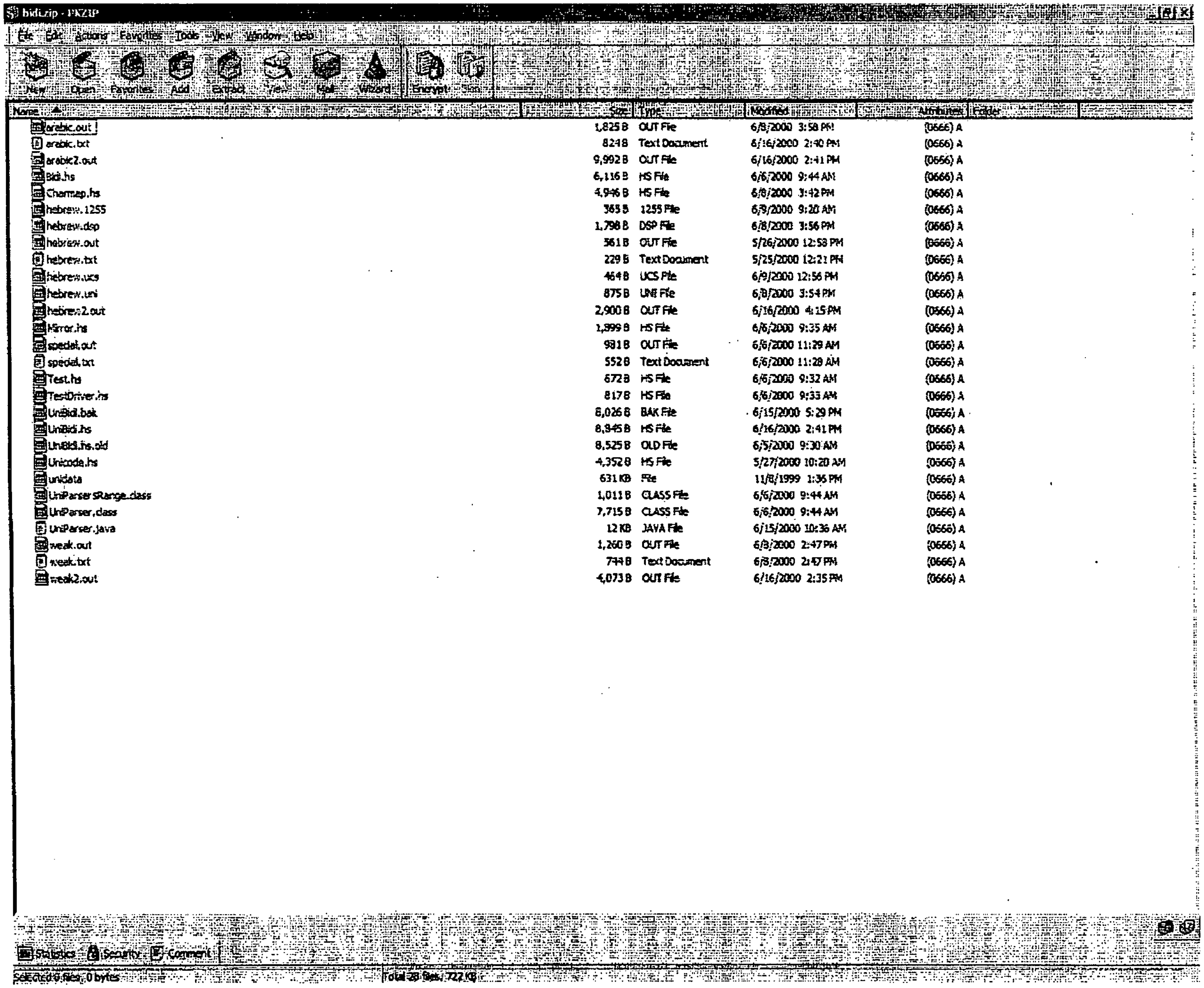
Sworn and subscribed to before me this _______ day of __________________, 2005.

______________________

Notary Public

My Commission Expires ______________________

Application No. 09/838,377 **AFFIDAVIT APPENDIX** Page 3 of 10


bidi.zip - PKZIP
File Edit Actions Favorites Tools View Window Help
New Open Favorites Add Extract View Mail Wizard Encrypt Sign
Statistics Security Comment

| Name | Size | Type | Modified | Attributes | Folder |
|---|---|---|---|---|---|
| arabic.out | 1,825 B | OUT File | 6/8/2000 3:58 PM | (0666) A | |
| arabic.txt | 824 B | Text Document | 6/16/2000 2:40 PM | (0666) A | |
| arabic2.out | 9,992 B | OUT File | 6/16/2000 2:41 PM | (0666) A | |
| Bidi.hs | 6,116 B | HS File | 6/6/2000 9:44 AM | (0666) A | |
| Charmap.hs | 4,946 B | HS File | 6/8/2000 3:42 PM | (0666) A | |
| hebrew.1255 | 365 B | 1255 File | 6/9/2000 9:20 AM | (0666) A | |
| hebrew.dsp | 1,798 B | DSP File | 6/8/2000 3:56 PM | (0666) A | |
| hebrew.out | 561 B | OUT File | 5/26/2000 12:58 PM | (0666) A | |
| hebrew.txt | 229 B | Text Document | 5/25/2000 12:21 PM | (0666) A | |
| hebrew.ucs | 464 B | UCS File | 6/9/2000 12:56 PM | (0666) A | |
| hebrew.uni | 875 B | UNI File | 6/8/2000 3:54 PM | (0666) A | |
| hebrew2.out | 2,900 B | OUT File | 6/16/2000 4:15 PM | (0666) A | |
| Mirror.hs | 1,899 B | HS File | 6/6/2000 9:35 AM | (0666) A | |
| special.out | 981 B | OUT File | 6/6/2000 11:29 AM | (0666) A | |
| special.txt | 552 B | Text Document | 6/6/2000 11:28 AM | (0666) A | |
| Test.hs | 672 B | HS File | 6/6/2000 9:32 AM | (0666) A | |
| TestDriver.hs | 817 B | HS File | 6/6/2000 9:33 AM | (0666) A | |
| UniBidi.bak | 8,026 B | BAK File | 6/15/2000 5:29 PM | (0666) A | |
| UniBidi.hs | 8,845 B | HS File | 6/16/2000 2:41 PM | (0666) A | |
| UniBidi.hs.old | 8,525 B | OLD File | 6/5/2000 9:30 AM | (0666) A | |
| Unicode.hs | 4,352 B | HS File | 5/27/2000 10:20 AM | (0666) A | |
| unidata | 631 KB | File | 11/8/1999 1:36 PM | (0666) A | |
| UniParser$Range.class | 1,011 B | CLASS File | 6/6/2000 9:44 AM | (0666) A | |
| UniParser.class | 7,715 B | CLASS File | 6/6/2000 9:44 AM | (0666) A | |
| UniParser.java | 12 KB | JAVA File | 6/15/2000 10:36 AM | (0666) A | |
| weak.out | 1,260 B | OUT File | 6/8/2000 2:47 PM | (0666) A | |
| weak.txt | 744 B | Text Document | 6/8/2000 2:47 PM | (0666) A | |
| weak2.out | 4,073 B | OUT File | 6/16/2000 2:35 PM | (0666) A | |

Selected 0 files, 0 bytes Total 28 files, 722 KB

**PRINT OUT OF UniParser.java, file dated 6/15/2000**

```
==========================================================================
/*  UniParser.java
 *  Steven Atkin 4/26/2000
 *  This class provides methods for parsing the Unicode data table published
 *  by the Unicode consortium www.unicode.org.
 */

import java.io.*;
import java.lang.*;
import java.text.*;
import java.util.*;
import javax.swing.*;

/**
 * UniParser parses the Unicode data table for a specific attribute
 * and then generates a Haskell 98 (Hugs 98) module for accessing the
 * attribute information.
 */
public class UniParser {
    public class Range{
        int start, end;

        public int getStart() {
           return start;
        }
        public int getEnd() {
           return end;
        }
        public void setStart(int begin) {
            start = begin;
        }
        public void setEnd(int last) {
            end = last;
        }
        public String toString() {
            return ("(0x" +
                    Integer.toHexString(start) +
                    ",0x" +
                    Integer.toHexString(end) +
                    ")");
        }
    }

    int unicodeField = 0, attributeField = 0;
    BufferedReader dataIn;
    BufferedWriter dataOut;
    String delim = ";";
    String moduleName, typeName, functionName;
    String attributes[];
    String defaultAttrib;
    Vector tables[];

    /**
      * Constructor - must specify input data file and
      * output module name.
      */
    public UniParser(String in, String out) {
        try {
            dataIn = new BufferedReader (new FileReader(in));
            FileOutputStream fileOut = new FileOutputStream (out);
            OutputStreamWriter str = new OutputStreamWriter (fileOut,
"windows-1252");
            dataOut = new BufferedWriter (str);
        }
        catch (UnsupportedEncodingException e1) {
            System.out.println(e1.toString());
            return;
        }
        catch (IOException e) {
            System.out.println(e.toString());
            return;
        }
    }

    /**
      * Parse the individual line for specified token
      */
    private String getToken(String data, int n) {
        StringTokenizer parser;
        String token = "";
        parser = new StringTokenizer(data, delim);
        while(parser.hasMoreTokens() && n > 0) {
            token = parser.nextToken();
            --n;
        }
        return token;
    }

    /**
      * Determine the Unicode codepoint value
      */
    private String getCodePoint(String data) {
        return getToken(data, unicodeField);
    }

    /**
      * Get the attribute field
      */
    private String getAttribute(String data) {
        return getToken(data, attributeField);
    }

    private void insert(Range r, String bi) {
        /* Add the codepoint to the corresponding vector */
            for(int i = 0; i < attributes.length; ++ i) {
                if(bi.equals(attributes[i])) {
                    tables[i].addElement(r);
                    break;
                }
           }
    }

    /**
      * Parse the entire data file for tokens
      */
    public void parse() {
        String in, cp, bidiattr, lastattr = "";
        int unival, lastunival = 0;
        Range range = new Range();
        StringTokenizer parser;
        try {
            /* Read one line at a time */
            while((in = dataIn.readLine()) != null) {
                cp = getCodePoint(in);
                unival = Integer.parseInt(cp, 16);
                /* Get the attribute value */
                bidiattr = getAttribute(in).toLowerCase();

                if(unival == 0) {
                        lastattr = bidiattr;
                        range = new Range();
                        range.setStart(0);
                }

                if(unival == 0xfffd) {
                        range.setEnd(unival);
                        insert(range, lastattr);
                }

                if(!bidiattr.equals(lastattr)) {
                        range.setEnd(lastunival);
                        insert(range, lastattr);
                        range = new Range();
                        range.setStart(unival);
                        lastattr = bidiattr;
                }
                lastunival = unival;
            }
            dataIn.close();
        }
        catch (IOException e) {
            System.out.println("error parsing data file");
            return;
        }
    }

    /**
     * Turn the vector's contents into a string
     */
    private String toString(Vector v) {
        int max = v.size() - 1;
        StringBuffer buf = new StringBuffer();
        Enumeration e = v.elements();
        buf.append("[");

        for (int i = 0 ; i <= max ; i++) {
           String s = e.nextElement().toString();
           buf.append(s);
           if (i < max)
           buf.append(", ");
            /* Write four per line */
            if ((i+1)%4 == 0 && (i < max))
               buf.append("\n    ");
        }
        buf.append("]");
        return buf.toString();
    }

    /**
```

```
         * Write the Haskell module header
         */
        private void writeModuleHeader() {
            Date date = new Date();
            DateFormat df = DateFormat.getDateTimeInstance(
                                DateFormat.FULL, DateFormat.FULL);
            try {
                dataOut.write("-- This file is computer generated do not modify");
                dataOut.newLine();
                dataOut.write("-- This is a Haskell 98 script file for Hugs98");
                dataOut.newLine();
                dataOut.write("-- " + moduleName + ".hs created on " + df.format
(date));
                dataOut.write(" by java UniParser");
                dataOut.newLine();
                dataOut.write("-- This module provides functions for obtaining " +
                              "Unicode character attributes");
                dataOut.newLine();
                dataOut.write("module " + moduleName + " (");
                dataOut.newLine();
                dataOut.write("    " + functionName + ", ");
                dataOut.newLine();
                dataOut.write("    " + typeName + ",");
                dataOut.newLine();


                for(int i = 0; i < attributes.length; ++i) {
                    dataOut.write("    " + attributes[i].toUpperCase());
                    if(i != attributes.length - 1)
                        dataOut.write(",");
                    dataOut.newLine();
                }
                /* Write the default type */
                if(defaultAttrib.length() != 0)
                    dataOut.write("    , " + defaultAttrib.toUpperCase());
                dataOut.newLine();

                dataOut.write(")" + " where");
                dataOut.newLine();
            }
            catch (IOException e) {
                System.out.println("error writing module header");
                return;
            }
        }

        /**
         * Create the attribute type for the haskell module
         */
        private void writeModuleTypes() {
            try{
                dataOut.write("data " + typeName + " = ");
                dataOut.newLine();

                /* Write each attribute as a type constructor
                 * with no arguments */

                for(int i = 0; i < attributes.length; ++i) {
                    dataOut.write("    " + attributes[i].toUpperCase());
                    if(i != attributes.length - 1)
                    dataOut.write(" |");
                dataOut.newLine();
            }
            /* Write the default type */
            if(defaultAttrib.length() != 0)
                dataOut.write("    | " + defaultAttrib.toUpperCase());
            dataOut.newLine();
            dataOut.write("    deriving(Eq, Show)");
            dataOut.newLine();
        }
        catch (IOException e) {
            System.out.println("error writing module types");
            return;
        }
    }

    /**
     * Write the test function for determing the attribute
     * of a Unicode character.
     */
    private void writeModuleAccessFunc() {
        try {
            dataOut.newLine();
            dataOut.write("-- (Start range, End range)");
            dataOut.newLine();
            dataOut.write("member :: Integer -> [(Integer, Integer)] ->
Bool");
            dataOut.newLine();
            dataOut.write("member a [] = False");
            dataOut.newLine();
            dataOut.write("member a ((x,y):xs)");
            dataOut.newLine();
            dataOut.write("    | a < x");
            dataOut.newLine();
            dataOut.write("      = False");
            dataOut.newLine();
            dataOut.write("    | a <= y");
            dataOut.newLine();
            dataOut.write("      = True");
            dataOut.newLine();
            dataOut.write("    | otherwise");
            dataOut.newLine();
            dataOut.write("      = member a xs");
            dataOut.newLine();
            dataOut.newLine();
            dataOut.write(functionName + " :: Int -> " + typeName);
            dataOut.newLine();
            dataOut.write(functionName + " x");
            dataOut.newLine();
            /* Check for all attributes */
            for(int i = 0; i < attributes.length; ++i) {
                dataOut.write("    | member (toInteger x) " + attributes[i]);
                dataOut.newLine();
                dataOut.write("      = " + attributes[i].toUpperCase());
                dataOut.newLine();
            }
            /* Check for default attribute */
            if (defaultAttrib.length() != 0) {
                dataOut.write("    | otherwise");
                dataOut.newLine();
                dataOut.write("     = " + defaultAttrib.toUpperCase());
                dataOut.newLine();
            }
        }
        catch (IOException e) {
            System.out.println("error writing module accessor function");
            return;
        }
    }

    /**
     * Write each vector out as a table.
     */
    private void writeModuleTables() {
        try {
            for(int i = 0; i < attributes.length; ++i) {
                dataOut.write(attributes[i] + " = " + toString(tables[i]));
                dataOut.newLine();
            }
        }
        catch (IOException e) {
            System.out.println("error writing module tables");
        }
    }

    /* Set the name of the Haskell module */
    public void setModuleName(String name) {
        moduleName = name;
    }

    /* Set the data type name for the Haskell module */
    public void setTypeName(String name) {
        typeName = name;
    }

    /* Set the name of the test function for the Haskell module */
    public void setAccessFuncName(String name) {
        functionName = name;
    }

    /* Set which column the Unicode field is in */
    public void setUnicodeField(int num) {
        unicodeField = num;
    }

    /* Set which field to parse */
    public void setParseField(int num) {
        attributeField = num;
    }

    /* Specify the list of attributes to parse for */
    public void setAttributeArray(String[] attribs) {
        attributes = attribs;
        tables = new Vector[attributes.length];
        for(int i = 0; i < tables.length ; ++ i) {
            tables[i] = new Vector();
        }
    }

    /* Specify the default attribute */
    public void setDefaultAttribute(String name) {
        defaultAttrib = name;
    }

    /**
     * Write the Haskell module file.
     */
    public void writeModule() {
        writeModuleHeader();
        writeModuleTypes();
        writeModuleTables();
        writeModuleAccessFunc();
        try {
            dataOut.close();
        }
        catch (IOException e) {
            System.out.println("error closing module file");
            return;
        }
    }

    /* Driver */
    public static void main(String[] args) {
        String attributes[] = {"r", "al", "en", "es", "et", "an", "cs",
                               "nsm", "bn", "b", "s", "ws", "on", "lre",
                               "rle", "pdf", "lro", "rlo", "n"};
        UniParser p = new UniParser("unidata", "Bidi.hs");
        p.setModuleName("Bidi");
        p.setTypeName("Bidi");
        p.setAccessFuncName("getBidiAttr");
        p.setUnicodeField(1);
        p.setParseField(5);
        p.setAttributeArray(attributes);
        p.setDefaultAttribute("l");
        p.parse();
        p.writeModule();
    }
}
```